# Knowledge representation

# lecture 5

# rdfs, rules and logic

Tanel Tammet
TTU

# Lecture overview

RDFS

RDFs and logic

OWL

OWL and logic

# RDFS

RDFS: RDF Schema

Three kinds of simple taxonomy rules added to RDF

„if X is a car, X is a vehicle“

„if X has a property profession, X is a person“

„if X has a property brother, value of the property is a person“

# First (main) rule

example:

ex:MotorVehicle rdf:type rdfs:Class

exthings:companyCar rdf:type ex:Van

ex:Van rdfs:subClassOf ex:MotorVehicle

# Main rule

same facts in the predicate calculus notation:

rdf:type(ex:MotorVehicle, rdfs:Class)

rdf:type(exthings:companyCar,ex:Van)

rdfs:subClassOf(ex:Van, ex:MotorVehicle)

built-in rule assumed in rdfs:

rdf:type(X,Y) & rdfs:subClassOf(Y,Z) => rdf:type(X,Z)

# Second rule

ex:Person rdf:type rdfs:Class .
ex:author rdf:type rdf:Property .
ex:author rdfs:range ex:Person

and the fact

ex:person1 ex:author ex:hamlet

should derive:

ex:author rdf:type ex:Person

# Logically ...

built-in rule assumed in rdfs:

Y(X,Z) & rdfs:range(Y,U) => rdf:type(X,U)

# Third rule

ex:Book rdf:type rdfs:Class .
ex:author rdf:type rdf:Property .
ex:author rdfs:domain ex:Book .

and the fact

ex:person1 ex:author ex:hamlet

should derive:

ex:hamlet rdf:type ex:Book

# Logically ...

built-in rule assumed in rdfs:

Y(X,Z) & rdfs:domain(Y,U) => rdf:type(Z,U)

# Additional encoding layer!

built-in rule assumed in rdfs:

rdf:type(X,Y) & rdfs:subClassOf(Y,Z) => rdf:type(X,Z)
Y(X,Z) & rdfs:range(Y,U) => rdf:type(X,U)
Y(X,Z) & rdfs:domain(Y,U) => rdf:type(Z,U)

have to be encoded in classical 1st order logic as

rdf(X,rdf:type,Y) & rdf(Y,rdfs:subClassOf,Z) => rdf(X,rdf:type,Z)
rdf(X,Y,Z) & rdf(Y,rdfs:range,U) => rdf(X,rdf:type,U)
rdf(X,Y,Z) & rdf(Y,rdfs:domain,U) => rdf(Z,rdf:type,U)

# Reification

Example:

- Fact: http://xx#121 ex:eesnimi "Jaan"
- Metainfo about the fact:
  timestamp: 10jaan2008,
  sisestaja:peeter,
  usaldusvaarsus:0.9

Example encoded as triples (invent object "13"):

13 rdf:object http://xx#121
13 rdf:predicate ex:eesnimi
13 rdf:subject "Jaan"
13 ex:timestamp 10jaan2008
13 sisestaja peeter

# Reification rule

built-in rule assumed in rdf:

rdf(X,rdf:subject,Y) &
rdf(X,rdf:predicate,Z) &
rdf(X,rdf:object,U)

=>

rdf(Y,Z,U).

# NOT provided in rdfs:

- cardinality constraints on properties, e.g., that a Person has exactly one biological father.
- specifying that a given property (such as ex:hasAncestor) is transitive, e.g., that if A ex:hasAncestor B, and B ex:hasAncestor C, then
  A ex:hasAncestor C.
- specifying that a given property is a unique identifier (or key) for instances of a particular class.
- specifying that two different classes (having different URIrefs) actually represent the same class.
- specifying that two different instances (having different URIrefs) actually represent the same individual.

# NOT provided in rdfs:

- specifying constraints on the range or cardinality of a property that depend on the class of resource to which a property is applied, e.g., being able to say that

  - for a soccer team the ex:hasPlayers property has 11 values,
  - for a basketball team the ex:hasPlayers property has 5 values.

- the ability to describe new classes in terms of combinations (e.g., unions and intersections) of other classes, or to say that two classes are disjoint (i.e., that no resource is an instance of both classes).

# Ways to extend rdfs

Two main approaches:

- Using traditional rules.
  - Current mainstream:
    RIF (Rule Interchange Format).
  - Older favorite:
    RuleML (Rule Meta-Language)
  - A complex evolving standard:
    CL (Common Logic)
- Using a specialised description-logic based language
  OWL (Web Ontology Language)